



Mamba Image Library Tutorial

Author:
Nicolas BEUCHER

August 23, 2009

Contents

1	Introduction	3
2	Why/When use Mamba ?	3
3	License	3
4	Requirements	4
5	Installation	4
5.1	On Windows	4
5.2	On Linux	4
6	Contents of the library	4
6.1	mambaCore.so/mambaCore.pyd	4
6.2	mambaCore.py	4
6.3	mambaBase.py	4
6.4	mambaError.py	5
6.5	mamba.py	5
6.6	mambaDraw.py	5
6.7	mambaExtra.py	5
6.8	package mambaComposed	5
6.9	package mambaRealtime	5
7	Using Mamba	5
7.1	A simple example	5
7.2	Importing the module	6
7.3	Grid and border	6
7.4	Windowing computations	7
7.5	Creating and manipulating images	8
7.6	Color palette	9
7.7	Displaying the image	10
7.8	Functions and classes	11
7.9	Modules in mambaComposed	11
7.10	More advanced examples	12
7.10.1	example 1	12
7.10.2	example 2	13
8	Drawings and Extra	14
9	Mamba realtime	15
10	Limitations and restrictions	15

List of Figures

1	Hexagonal grid	6
2	Square grid	6
3	Directions on hexagonal grid	7
4	Directions on square grid	7
5	Effects of the rainbow and inverted rainbow palettes on the first image	10
6	Image displayer window	10
7	Labeling, input and result	12
8	Distance to set border, input and result	14

1 Introduction

This document is a tutorial explaining the basics of the library for Python *Mamba Image*.

Mamba Image actually stands for *M*athematical *M*orphological *liBr*ary *I*mage (For sake of simplicity, the library will be referred to as Mamba). The library provides a set of functions needed to perform basic operations used in mathematical morphology like erosion, dilation, etc...

Before reading this document, make sure you have basic knowledge of Python programming language (see the Python tutorial at <http://docs.python.org/tutorial/>) and that you know the basics of mathematical morphology (online courses are available at <http://cmm.ensmp.fr/~serra/acours.htm>).

Mamba is written partly in C for low level functions and in Python for high level interface and image handling.

2 Why/When use Mamba ?

Before going further in this tutorial, let's see why you should use mamba and, almost more importantly, when. It can be sum up is the following sentence :

Mamba is meant to be a fast and easy library for coding mathematical morphology algorithms.

What we meant here is that, if your main concern is to try new algorithms to address your morphology mathematical problem while not waiting all night long for your result to come out or worse not come out because you made a mistake, then Mamba is meant for you. However, if you are looking for a library to perform image processing tasks like convolutions, contrast enhancers or likewise, you had better not using it (try PIL instead). And by the way, Mamba does not make coffee (sorry about that...).

To do so, Mamba low level library is coded in C with performance through simplicity in mind. The Python wrapper purpose is to give you an interface to that code that is fast to code and easy to play with (no compilation required and interactive help). Another objective regarding Mamba code is to be as portable as possible. This means that if you need to port your algorithm to a specific system you can easily adapt the C code to it.

Regarding licensing, Mamba is released under X11 license (also known as MIT license), see section 3 for more information.

To conclude this, let me remind you that mambas are fast-moving land-dwelling snakes of Africa. Their bite (at least for the black mamba) is extremely deadly but we assure you that no harm may come to you using Mamba... Well it won't bite you.

3 License

Here is a copy of the license of Mamba. This license is known as the X11 license (also named MIT license).

Copyright (c) <2009>, <Nicolas BEUCHER and ARMINES for the Centre de Morphologie Mathématique(CMM), common research center to ARMINES and MINES Paristech>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Except as contained in this notice, the names of the above copyright holders shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without their prior written authorization.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

4 Requirements

To use Mamba, you will need :

- Python version 2.6 or earlier.
- Python Imaging library (PIL) for your current version of Python.
- Tkinter (normally comes with Python on Windows systems but you may need to install it on UNIX systems).

To compile the sources, you will need :

- Python version 2.6 or earlier with the distutils package.
- Swig version 1.3.33 or earlier.
- GCC version 4.3.0 or earlier (or its Windows version MingW32)

Compilation instructions are given in the README file provided with the library.

Mamba works with all kind of processors. The library takes also profit of 64-bit processors. To enable the 64-bit mode, you will need a 64-bit Python installed on your computer.

5 Installation

5.1 On Windows

Download the Windows installer corresponding to your version of Python and run it. Follow the instructions.

5.2 On Linux

Use distutils package to compile and install the Mamba source.

6 Contents of the library

Before going any further, let us have a look at what is installed with the library.

All the files can be found in the site-packages/ directory of your local Python installation under the directory mambaIm.

6.1 mambaCore.so/mambaCore.pyd

This is the main C library compile for python (mambaCore.so on Linux systems and mambaCore.pyd on Windows systems). This file contains all the basic functions and image structures definitions used by Mamba to perform the various mathematical morphology operations.

6.2 mambaCore.py

This file is automatically created by SWIG. It is the direct interface to the C library.

6.3 mambaBase.py

This is the basic module of the library whose main purpose is to produce a more advanced layer to wrap C functions. It provides functions to load, create, compute and change options for the library. It also provides basic constants and library information.

This file provides a Python wrapper for each basic C function found in mambaCore.so.

This module is not user friendly and is not meant to be use directly so it is not advised to program using it (except if you think you know what you are doing).

6.4 `mambaError.py`

This module provides error handling functions for Mamba. C functions return an error code that can be interpreted using this module. Exceptions are raised if an error occurred. This module will most likely never be used by a regular programmer.

6.5 `mamba.py`

This is the main module of the Python Mamba library. It provides classes, functions and utilities (graphical interface) to perform computations in a simple and user-friendly environment. This module uses Tkinter to create windows displaying each image that is processed. Thus it allows the developer to see rapidly and efficiently the results of his work.

The Mamba module has been designed to facilitate the development and debugging of your program. The performance is known to be lowered due to display update. However it can be disabled and enabled at will allowing to suppress the performance overhead.

This tutorial will only explain how this module works and what you can do with it. Its full content will be detailed in section 7.

Before you go complaining, you should know that this module does not provide a complete graphical interface (GUI) to control all the parameters. You still have to use the command line a lot...

6.6 `mambaDraw.py`

This module provides a set of functions to draw inside the image. You can draw lines, squares,...

6.7 `mambaExtra.py`

This module provides a set of functions to perform specific display. They are here to help the programmer understand its results.

6.8 package `mambaComposed`

This package provides you with a set of modules containing basic (and less basic...) functions of mathematical morphology. A complete list of the modules is given at the end of this tutorial. The functions all use Mamba as a base. `mambaComposed` can then be seen as an extension to Mamba.

6.9 package `mambaRealtime`

Currently only working on Linux systems, this package provides an interface to acquire, process and display a video from any Video4Linux (V4L) or Video4Linux2 (V4L2) compatible acquisition devices.

7 Using Mamba

7.1 A simple example

Before anything else, a little example :

```
#A simple test for mamba
from mamba import *
from mambaComposed.erosion import erodeInAllDirImage as erosion

im1 = imageMb('cat.bmp', depth=1)
im2 = imageMb(depth=1)
erosion(im2, im1)
```

This code performs an erosion operation on image `im1` which contains the binary image 'cat.bmp'. The first line of code is to import the Mamba module completely thus allowing to use directly its functions and classes. The second line allows to import the `erodeInAllDirImage` function from the `mamba.Composed.erosion` module and to rename it as `erosion`. The third line tells Python that we want to create a binary image (1 pixel per bit), referred to as `im1`, using 'cat.bmp' as the source for pixel values. The fourth line tells Python to create an empty (i.e black) binary image referred to as `im2`. The last line actually performs an erosion of `im1` and put the result in `im2`.

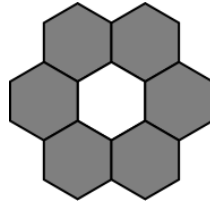


Figure 1: Hexagonal grid

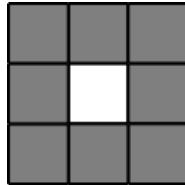


Figure 2: Square grid

Note, in this example, the order of the `im2` (destination) and `im1` (source) images. The destination must be put before the source, because the erosion function is defined this way.

Of course you can perform this example directly on the Python command line or create a Python script and run it. However in the last case, you will not be able to see the result as Python will close once it reaches the end of your file. To see the result you could do the following :

```
#A simple test for mamba
from mamba import *
from mambaComposed.erosion import erodeInAllDirImage as erosion

im1 = imageMb('cat.bmp', depth=1)
im2 = imageMb(depth=1)
erosion(im2, im1)
#To display the result
im2.showDisplay()
#To save your result inside a bmp file
im2.save('eroded_cat.bmp')
```

7.2 Importing the module

To use Mamba, simply type in your Python console:

```
import mamba

or

from mamba import *
```

The later will give you direct access to the module functions and variables.

7.3 Grid and border

Grid and border are important notions in mathematical morphology. So, before doing anything else, we will explain how they are handled in Mamba.

The grid defines the way pixels interact with their neighbours. There is two possible grids in Mamba: **hexagonal** or **square**. The **hexagonal** grid defines six neighbours for each pixel as can be seen in figure 1. The **square** grid defines the usual eight neighbours for each pixel as in figure 2.

The border defines the behaviour of all the pixels that are not in the image. An image is a finite set of pixels, however morphological mathematical operations assume an infinite world. Consequently, to be able to compute them, we have to virtualize the infinite world. The border can be set to **empty** or **filled**. The **empty** border is assuming external world is empty (the border is set to 0) whereas the **filled** border assumes an external world completely filled (the border is set to the maximum possible value).

By default, Mamba uses the **hexagonal** grid and the **empty** border. Mamba provides functions to change those parameters. The functions also ensure that the operations who are affected by the grid or border status are

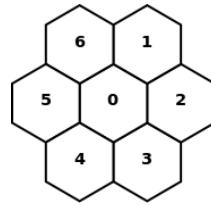


Figure 3: Directions on hexagonal grid



Figure 4: Directions on square grid

correctly mapped to the corresponding functions (e.g. the function performing the erosion is different whether you use an hexagonal or a square grid). Please note that some functions available in the `mambaComposed` package may change the grid and border settings for the length of their computations.

For the grid, you can use the three following functions :

```
# Prints the current grid status
printGrid()

# Returns the current grid value (to be able to store it)
getGrid()

# Changes the current grid value to HEXAGONAL
setGrid(HEXAGONAL)

# Changes the current grid value to SQUARE
setGrid(SQUARE)
```

For the border you can use the three following function :

```
# Prints the current border status
printBorder()

# Returns the current border value (to be able to store it)
getBorder()

# Changes the current border value to EMPTY
setBorder(EMPTY)

# Changes the current border value to FILLED
setBorder(FILLED)
```

As you will see later in this document, some functions need a direction as input. Figures 3 and 4 give you the direction encoding depending on the grid in use. The directions are numbered from 0 to 6 or 8.

You can get a list of all the available directions given the grid that is in use by calling the function `getDirections`.

```
# Returns a list of all the available directions in the current grid
directions = getDirections()
```

7.4 Windowing computations

In some case, you will not need or want to proceed the whole image, but only a part of it. In these cases, you will need to use the windowing functions of Mamba to indicate the region of the image in which you want to proceed.

```
# Only the window defined by upper left pixel (50,46) and lower right pixel
# (100,126) will be processed (pixel included in the window)
setWindow((50,46,129,115))
```

In the example above, the window given in argument is not valid for Mamba low level functions which require a window size with a width that is a multiple of 64 and a height that is a multiple of 2. Similarly, the window cannot start anywhere but at an abscissa that is a multiple of 64. The window defined in the example will then result in a computation window of (0,46,191,116). You can have information regarding the current image size, window size and position by using the following functions:

```
# Getting the window size and position
getWindowSize()
getWindowPosition()
# Getting the image size (internal to Mamba and as originally given)
getImageSize()
getOriginalImagePosition()
```

Beware of the functions depending on the border value for their computations. Every pixel outside the window is considered as part of the border, even if the pixel belongs to the image.

Pixel manipulation functions are not affected by the window, the pixel position remains absolute.

Finally if you want to come back to process the whole image just use the reset function :

```
# Resetting the window
resetWindow()
```

7.5 Creating and manipulating images

Creating images To handle images, a Python class has been created. `imageMb` will allow you to create, load, save and perform standard operations on your images.

As examples are better than a long explanation, let's have a look at these :

```
# Loading image 'cat.bmp' inside a binary image
im1 = imageMb('cat.bmp', 1)

# Loading image 'lion.PNG' inside a grey scale image
im2 = imageMb('lion.PNG')

# Creating an empty grey scale image
im3 = imageMb()

# Creating an empty binary image
im4 = imageMb(1)
# or alternatively
im4 = imageMb(depth=1)

# Creating an empty 32 bit image
im4 = imageMb(32)
# or alternatively
im5 = imageMb(depth=32)

# Loading image 'lion.PNG' red component inside a grey scale image
im6 = imageMb('lion.PNG', rgbfilter=(1.0,0.0,0.0))
```

In the previous examples, we declared each variable `imx` to be an `imageMb` object. The constructor can take two arguments that are optional: a path to an image file and a depth value (as was done for `im1`). Those arguments can be omitted. An empty grey scale image will be created by default (as in `im3`). You can give the path without indicating a depth (as in `im2`) and it will create a grey scale image filled with the pixel values of your image (color image are converted automatically). You can also give only the depth to create an empty image (filled with 0) in the required format (as in `im4` and `im5`, although in this case you have to specify that its the depth argument that you are providing as Python expects the path in the first place). Eventually, you can load a specific color component (or a specific mix of the color components) using the option `rgbfilter` (as in `im6`). The option takes a tuple with three float values (0 to 1) giving the amount of red, green and blue respectively to take from the image (the option is used only when loading an image).

Mamba supports all the image formats that are supported by PIL (including png, jpeg, bmp, gif, ...). You can create binary images (`depth=1`), grey scale images (`depth=8`) or 32-bit signed images (`depth=32`).

Saving and loading images Once you have created an image and performed some operations on it you may want to save it into a file. The `imageMb` class provides the method `save()` to do so :

```
# Saving the content of im1 image in test.png
im1.save('test.png')
```

The method uses PIL functions to save the image. Thus the format (bmp, gif, jpeg, ...) is automatically deduced from the extension used.

If you want to load an image into your `imageMb` object (assuming you did not make it at the image creation) you can use the method `load()` to do so :

```
# Loading the content of test.png in image im1
im1.load('test.png')
```

Other imageMb methods Besides creating, loading or saving your image, a range of other functionalities are accessible using specific methods.

If you need to convert an image into another format (another depth), use the `convert()` method. Supported conversions are binary to grey scale (1->8) or grey scale to binary (8->1). A binary image is converted to a grey scale image using 0 for false and 255 for true. A greyscale image is converted to binary using the following method, every pixel that is equal to 255 is set to true, otherwise false. `convert()` takes the required depth as argument. This method is here for convenient purposes, there are more elaborated ways to convert image provided inside Mamba.

```
# Converts the greyscale image into binary format
greyscale = imageMb(depth=8)
greyscale.convert(1)
```

```
# Converts the binary image into greyscale format
binary = imageMb(depth=1)
binary.convert(8)
```

If you want to erase an image, you can use the `fill()` method. It allows you to set all the pixels of an image to a given value.

```
# Erase the image im1 by setting all its pixels to 0
im1.fill(0)
```

```
# Fill im2 image with the value 3
im2.fill(3)
```

Although displaying the image is very efficient to follow the progress of an algorithm, it may sometimes get confusing as more and more images fill the screen. Moreover, grey-scale images are not always very readable. To enhance your display management, you can use the following methods :

```
# Indicates the name of the image to be displayed in window
# title
im1.setName('main image')
```

```
# Indicates that im1 will use palette as a color conversion
# table for display
im1.setPalette(palette)
```

```
# Reset any palette conversion (the image will return to greyscale display)
im1.resetPalette()
```

7.6 Color palette

Color palette defines the way a grey pixel will be converted to color. It associates for each possible value (0-255) a tuple of values describing the red, green and blue components (R, G, B) of the wanted color. The final palette is built by concatenating all this tuple in a single one.

Mamba comes with two predefined palettes :

```
# The rainbow color (0 equals black)
rainbow
```



Figure 5: Effects of the rainbow and inverted rainbow palettes on the first image

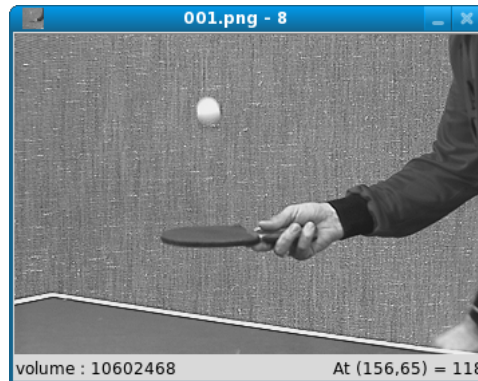


Figure 6: Image displayer window

```
#The inverted rainbow (0 equals black)
inverted_rainbow
```

You can see the results of using these palettes in figure 5.

When you apply a palette to an image, it will affect its display and the image stored by saving. The computations are not impacted. Be also aware that, if you save your image in color, you may have different values when reopening it with Mamba as the color to grey scale converter is not using your palette.

7.7 Displaying the image

When trying new algorithms, it can be very useful to be able to follow the evolution of your images without needing to save the image at each step. Mamba provides a graphical interface, see figure 6, which can display every required image into its own display window.

To activate a display, type :

```
# Show the image im in its own window
im.showDisplay ()
```

Every modification on this image will then be shown inside the opened window. This operation is known to be very demanding on performance. The display also gives information regarding the image volume, the position of the mouse inside the image (and the pixel value associated). You can zoom in and out using keys Z and A.

If you want to disable the display you have two options, either close the window by clicking the appropriate button or type :

```
# Hide the window associated with im
im.hideDisplay ()
```

Both operations will prevent the display to be updated for every modification of the image and thus will remove the performance overhead.

When the display is active, a set of functions will allow you to dynamically interact with it. These functions are called tracking functions. To activate the tracking for an image type :

```
# Activate the tracking for im display
im.enableTrack ()
```

Now every time you click inside the image, the event is stored inside a list of events that can be obtained using the appropriate function. There are two types of events: "PIXEL" and "SELECTION". The first one corresponds to a direct click inside the display (only one pixel is selected). The second is obtained when the mouse is moved while the button stay pressed (leading to the selection of a window inside the image).

```
# Obtain the list of mouse events inside the image
events = im.getTrack()

# events is a list of mouse events
print events
[['SELECTION', 412, 149, 789, 426], ['PIXEL', 394, 411]]

# The list is erased by the call to getTrack, so a consecutive call will return
# an empty list
events = im.getTrack()
print events
[]
```

The list contains mouse events where the first element indicates the type, either "PIXEL" or "SELECTION", and the following represents the position of the clicking ["PIXEL", x, y] or the selected window ["SELECTION", x1, y1, x2, y2] (there is no guarantee that x2 or y2 is greater than x1 or y1 respectively as the selection can be made backwards). The previously recorded event list is erased when this function is called again.

To disable the tracking, type :

```
# Deactivate the tracking for im display
im.disableTrack()
```

7.8 Functions and classes

More information regarding functions and classes that can be found in the Python library is available in the "Mamba Library Python Reference" document. Refer to it if you need help understanding a function or regarding possibilities.

Function names should be self explaining. Each function is detailed along with a small example to show you how to use it.

Of course, you can always use the Python interpreter help() function.

```
# Getting help over the list of functions
help(mamba)

# Getting help for a specific function (here addImages)
help(mamba.addImages)
```

7.9 Modules in mambaComposed

Here is the complete list of modules found in mambaComposed :

- erosion : defines functions to perform various erosions.
- dilation : defines functions to perform various dilations.
- build : defines functions to perform build operations.
- gradient : defines morphological gradient functions.
- openclose : defines closing and opening operators functions.
- filter : defines filtering functions such as alternate filters and top hats.
- sequence : defines functions and classes to work on images sequences.
- statistic : defines image statistic functions (mean, variance...).

To use them simply add this line in your Python script :

```
# example, importing erosion module functions
import mambaComposed.erosion

# Getting help and the list of functions
help(mambaComposed.erosion)
```

These modules define the most common functions used in mathematical morphology algorithms.

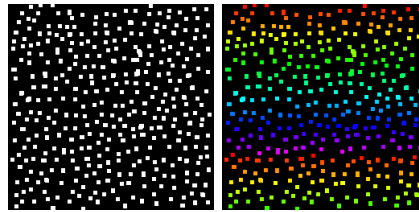


Figure 7: Labeling, input and result

7.10 More advanced examples

Now that you are more familiar with Mamba functions, let us see more advanced examples. You can use them as a basis to start your own programs. Each example is commented so that no external explanation should be required.

Other examples (simpler ones) can be found at <http://www.mamba-image.org>.

7.10.1 example 1

This file can be found in the Mamba source in `examples/` directory under the name `example1.py`. See the expected result for the image `im3` in figure 7.

```
from mamba import *

# The build function is created using the basic buildNeighborImage functions
# and the getDirections support function
# Usually you would use the build function provided in the mambaComposed.build
# module
def build(imInout , imIn):
    vol = 0
    prec_vol = -1
    while(prec_vol!=vol):
        prec_vol = vol
        for d in getDirections():
            vol = buildNeighborImage(imInout , imIn , d)

# A labeling function created with basic Mamba
# functions and not using the labelImage function
def label(dest , src):

    # We will need some temporary images
    # to perform computations
    provb = imageMb(depth=1)
    prov8 = imageMb()
    copy_src = imageMb(depth=1)

    # The source image is copied in a temporary
    # image for computations
    copyImage(copy_src , src)
    volume = -1
    label = 1

    # The while loop will be called until either the image is empty
    # which means that all the objects inside the image have been
    # processed or when the label index reach 256 which is the maximum
    # number of objects this function can label.
    while(not checkEmptinessImage(copy_src) and label < 256):
        # Erasing the image provb
        provb.fill(0)
        # Comparing provb to copy_src and putting the result
        # in provb. The result is that the first non black pixel
        # of copy_src is set in provb.
        compareImages(provb , provb , copy_src)
```

```

# Build of provb using copy_src as a mask
# this will build only the set attached to
# the pixel found by comparison.
build(provb, copy_src)
# Xor between provb and copy_src. the effect is to
# completely remove the set found by previous build
# from copy_src
logicImages(copy_src, copy_src, provb, 'xor')
# the set found in provb is converted to greyscale
# format using convertByMask to ensure that the value
# for True pixels is different for all sets
convertByMaskImage(prov8, provb, 0, label)
label = label+1
# The set is added to the destination image
addImages(dest, dest, prov8)

return label-1

# We will label the binary image 'bcp_obj.bmp'
im1 = imageMb('bcp_obj.bmp', 1)

# firstly, using the embedded labeling function
# 'labelImage'
im2 = imageMb(depth=32)
im3 = imageMb()
im4 = imageMb()
# some colors
im3.setPalette(rainbow)
im4.setPalette(rainbow)
im3.setName('labelImage 1')
im4.setName('labelImage 2')
print labelImage(im2, im1)
# The result is on 32-bit image so to be able to see it
# we copy it inside grey scale image byte plane by byte plane
copyBytePlaneImage(im3, im2, 0)
im3.showDisplay()
copyBytePlaneImage(im4, im2, 1)

# secondly, by the 'label' function created earlier
im5 = imageMb()
im5.setPalette(rainbow)
im5.setName('label')
print label(im5, im1)
im5.showDisplay()
# Here there is no need to extract byte plan as the
# result is already in grey-scale. However the slight defect
# is that this function cannot label more than 255 objects
# whereas the labelImage function can count up to 2 Billions
# which is unlikely to happen as Mamba cannot process image
# greater than 4000*4000.

```

7.10.2 example 2

This file can be found in the Mamba source in examples/ directory under the name example2.py. The result is shown in figure 8.

```

from mamba import *
# Here we will import the most common function of the mambaComposed module
# call erosion the erodeInAllDirImage function
from mambaComposed.erosion import erodeInAllDirImage as erosion
# renaming it in the process to make its use easier.

# A distance function using successive erosions
def distance(imOut, imIn):

```

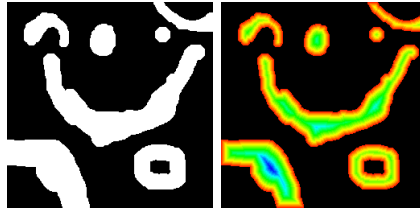


Figure 8: Distance to set border, input and result

```

volume = 1
imcp = imageMb(depth=1)
copyImage(imcp, imIn)

while(not checkEmptinessImage(imcp)):
    addImages(imOut, imOut, imcp)
    erosion(imcp, imcp)

# We will use a FILLED BORDER
setBorder(FILLED)

# we will compute the distance for image 'test_dist2.bmp'
im1 = imageMb('test_dist2.bmp', 1)

# First with the embedded function setBorderDistanceImage
im2 = imageMb(depth=32)
im3 = imageMb()
im3.setPalette(rainbow)
im3.setName('by setBorderDistance')
setBorderDistanceImage(im2, im1)
copyBytePlaneImage(im3, im2, 0)
im3.showDisplay()

# Secondly with the function created previously
im4 = imageMb()
im4.setName('by multiple erosion')
im4.setPalette(rainbow)
distance(im4, im1)
im4.showDisplay()

# We compare the two results
# the result must be (-1,-1)
im7 = imageMb()
im7.setName('comparison')
print compareImages(im7, im4, im3)

# To enhance the contrast we will multiply the images by
# a constant
mulconstImage(im3, im3, 8)
mulconstImage(im4, im4, 8)

```

8 Drawings and Extra

Two specific modules have been created to allow you to draw inside Mamba images and to offer some extra display methods.

The module `mambaDraw` offers functions to draw various forms inside the image, e.g. lines, squares or boxes, and functions to extract information regarding pixel values (intensity along a segment, ...).

The module `mambaExtra` offers functions to display, interact and generally perform operations that need the constant intervention of the programmer. For example, this module provides an interface to allow dynamic threshold of a given image (threshold value can then be modified on the fly by the user).

Both modules are described in the Python Reference document.

9 Mamba realtime

The realtime module is currently available only for Linux systems.

Basically it provides functions to open an acquisition device (webcam, tv set...), obtain image from it and display the result of some algorithm using mamba functions inside a fast display.

The module is documented in the Python Reference document.

10 Limitations and restrictions

Unfortunately, Mamba has some built-in limitations that may be corrected later but that you will have to deal with in the meantime.

Firstly, you cannot play with different image sizes at the same time. This is unsupported by the low-level C functions. Currently, it only raises a warning in the Python interface and crops or pads your image so that it fits with the current size you are playing with. If you want to change the size, you will have to restart Python.

Whatever image you load, it will always be assumed that it is a grey scale image even if it is a 1-bit depth bitmap image. Moreover, if there are some colors in your image, the grey-scale conversion may not be appropriate if you want to do computations only on the blue/red/green component.

Mamba is known to have some troubles with IDLE. Mamba uses Tkinter for image display and this may provoke interferences with IDLE. To prevent it, the recommended solution is to create a separate Python file where you put the Mamba import statement line. Then edit this file with IDLE and use the corresponding menu to run the module (shortcut is F5 by default).